

Lernzettel

Stacks: Implementierungen und Anwendungen

Universität: Technische Universität Berlin
Kurs/Modul: Einführung in die Informatik - Vertiefung
Erstellungsdatum: September 20, 2025



Zielorientierte Lerninhalte, kostenlos!
Entdecke zugeschnittene Materialien für deine Kurse:

<https://study.AllWeCanLearn.com>

Einführung in die Informatik - Vertiefung

Lernzettel: Stacks: Implementierungen und Anwendungen

(1) Grundidee.

Ein Stack ist ein abstrakter Datentyp, der das Last-In-First-Out-Prinzip (LIFO) implementiert. Alle Zugriffe erfolgen am oberen Stackrand. Die wichtigsten Operationen sind: `push`, `pop`, `peek`, `isEmpty` und `size`.

(2) Implementierungen.

Stacks lassen sich vor allem auf zwei Arten implementieren:

- **Array-basierter Stack.**

- Vorteile: schnelle Zugriffe, einfach umzusetzen, gut cache-friendly.
- Nachteile: feste oder dynamisch wachsende Kapazität; Resize-Strategie nötig.
- *Pseudocode*:

```
Push(x):  if size == capacity then resize(); data[size] = x; size++;
Pop():    if isEmpty() -> Fehler; size--; return data[size];
```

- **Verketteter Stack (Linked List).**

- Vorteile: dynamische Größe, kein Resize nötig; einfacher Speicherverbrauch pro Element.
- Nachteil: zusätzlicher Pointer pro Element.
- *Pseudocode*:

```
Push(x):  create neuer Knoten mit Wert x; neuerKnoten.next = top; top =
neuerKnoten;
Pop():    if isEmpty() -> Fehler; top = top.next; return Wert des alten
Tops;
```

(3) Operationen – Überblick.

- `Push(x)`: Element oben auf den Stack legen; $top \leftarrow x$.
- `Pop()`: oberstes Element entfernen und zurückgeben; `top` entsprechend anpassen.
- `Peek()`: oberstes Element zurückgeben, ohne zu entfernen.
- `IsEmpty()`: True, falls keine Elemente vorhanden sind.
- `Size()`: aktuelle Anzahl der Elemente im Stack.

(4) Komplexität.

- Push und Pop in einem Array-Stack: $\mathcal{O}(1)$ amortisiert.
- Peek: $\mathcal{O}(1)$.
- IsEmpty und Size: $\mathcal{O}(1)$.
- *Resize* eines array-basierten Stacks hat zeitliche Kosten, amortisiert sich aber über viele Operationen.

(5) Anwendungen.

- Ausdrucksbewertung und Umwandlung (Reverse-Polish-Notation, Infix zu Postfix).
- Balancierte Klammern prüfen (z. B. ([])).
- Tiefensuche (DFS) in Graphen, Backtracking-Algorithmen.
- Undo-/Redo-Mechanismen in Anwendungen.

(6) Balancierte Klammern prüfen – Beispiel.

Algorithmus (Textbeschreibung): Durchlaufe die Zeichenkette von links nach rechts. Wenn du eine öffnende Klammer (, [, { findest, push sie auf den Stack. Wenn eine schließende Klammer),], } auftaucht, prüfe, ob der Stack nicht leer ist und ob die oberste Öffnung dem passenden Typ entspricht. Ist er das, poppe; ist er es nicht oder der Stack ist leer, dann ist die Klammerung ungültig. Am Ende gilt: Stack muss leer sein.

Pseudocode (Beispiel in Java-ähnlicher Syntax):

```
for (char c : input)
    if (c in "(") push(c);
    else if (c in ")"]")
        if (isEmpty() || !matches(pop(), c)) return false;
    endfor
return isEmpty();
```

Hilfsfunktion: matches(open, close) prüft, ob die Paarung '(' , ')', '[' , ']', '' , '' stimmt.

(7) Beispiel-Implementierungen (Kernideen).

Array-basierter Stack (Java-ähnlich):

```
class Stacklt;Egt; {
    private E[] data;
    private int size;
    public void push(E x) { if (size == data.length) resize(); data[size++] = x; }
    public E pop() { if (isEmpty()) throw new EmptyStackException();
        E r = data[--size]; data[size] = null; return r; }
    public boolean isEmpty() { return size == 0; }
    private void resize() { /* neues Array; kopieren */
}
}
```

Verketteter Stack (Java-ähnlich):

```
class Node<E> { E value; Node<E> next; }
class Stack<E> { private Node<E> top;
public void push(E x) { top = new Node<E>(x, top); }
public E pop() { if (top == null) throw new EmptyStackException();
    E v = top.value; top = top.next; return v; }
public boolean isEmpty() { return top == null; }
}
```

Hinweis zur Typisierung: In Java-Templates können Generics verwendet werden (`Stack<E>`); bei konkreten Anwendungen kann man auch `Stack<Integer>`, `Stack<Character>`; usw. verwenden.